Tapestry5HowToCreateATabPanel

I wanted to create a tab panel in T5, and Kris Marinkovic was so nice to explain it to me in his email on June 11, 2007 (in the tapestry users email list). It's actually quite simple to do, and I've learned some new concepts from it too, so let's get to it. BTW If you fancy another approach, Todd has developed a component library with a tab panel. Check out his blog.

Assume you have created a Tab component (e.g. declared in org.example.myapp.components). I'll keep it simple, and this component's Tab.html isn't there (so there is no template), and the Tab.java only contains a default setter to display a message.

```
package org.example.myapp.components;

import org.apache.tapestry.MarkupWriter;
import org.apache.tapestry.annotations.BeginRender;

public class Tab {
        private String message="Bonjour from HelloWorld component.";

        @BeginRender
        void renderMessage(MarkupWriter writer) {
                writer.write(message);
        }

        public void setMessage(String message) {
                this.message = message;
        }
}
```

Now that you have a component to display the tab panel's content, you need a way to select it. Let's assume your Start page needs to contain the tab panel. To make it more interesting, I've used Yahoo's YUI library to create a header/footer/secondary/main page layout. So here is your Start.html template file:

```
<html xmlns:t="http://tapestry.apache.org/schema/tapestry_5_0_0.xsd">
<head>
<title>Start Page</title>
<!-- Load all necesary javascript files and style sheets -->
<link rel="stylesheet" href="css/reset-fonts-grids.css" type="text/css"></link>
<link rel="stylesheet" href="css/styles.css" type="text/css"></link>
</head>
<body>
<div id="doc" class="yui-t2">
<div id="hd">
<p align="center">
Header placeholder
</p>
<p align="right">
        <t:actionlink t:id="tab1_link" title="My first tab">Tab1</t:actionlink>
        <t:actionlink t:id="tab2_link" title="My second tab">Tab2</t:actionlink>
        <t:actionlink t:id="tab3_link" title="My third tab">Tab3</t:actionlink>
</p>
<hr/>
</div>

<div id="secondary" class="yui-b">
Menu placeholder
</div>

<div id="yui-main">
        <div class="yui-b">
                <div id="mainContent">
                        <t:body/>
                        <t:delegate to="selectedTab"/>
                        <t:block>
                                <div t:id="tab1"/>
                                <div t:id="tab2"/>
                                <div t:id="tab3"/>
                        </t:block>
                </div>
        </div>
</div>

<div id="ft">
<hr/>
<p align="center">Placeholder for footer</p>
</div>

</div>
</body>
</html>
```

Pay special attention to the `<t:actionlink t:id="tab1_link" ...`, which declares the tabs. Furthermore, the `<t:delegate to="selectedTab" />` instruction above is a placeholder for your new component. Since a delegate doesn't do any rendering (i.e. no output is generated), it requests the Start.java file to return a component which can render itself. This component can have the name tab1, tab2 or tab3, and replaces the respective `<div t:id="tab1"/>` in the `<t:block>` statement.

To complete the story, I'll show you the Start.java file:

```
package org.example.myapp.pages;

import org.apache.tapestry.annotations.Component;
import org.apache.tapestry.annotations.Persist;

import org.example.myapp.components.Tab;

public class Layout {
        // All components need to be declared (otherwise, your page will generate an error, since the <t:block>
block
        // expects them and names them.
        @Component private Tab tab1;
        @Component private Tab tab2;
        @Component private Tab tab3;

        // Due to the page redirect, you need to persist the value briefly
        @Persist("flash")
        private int selectedComponent;

        // These are the event methods that are generated by clicking on the Tab. Due to the specific
        // syntax, onActionFromIDNAME, there is no need for the @OnEvent annotation, although that would work
too.
        public void onActionFromTab1_link() {        selectedComponent = 1; }
        public void onActionFromTab2_link() {        selectedComponent = 2; }
        public void onActionFromTab3_link() {        selectedComponent = 3; }

        // Returns the selected tab, which will take the place of the corresponding <div t:id="tabXXX"/>
        public Object getSelectedTab() {
                switch (selectedComponent) {
                case 1:
                        tab1.setMessage("tab1");
                        return tab1;
                case 2:
                        tab2.setMessage("tab2");
                        return tab2;
                case 3:
                        tab3.setMessage("tab3");
                        return tab3;
                default:
                        tab1.setMessage("tab1");
                        return tab1;
                }
        }
}
```

That should do the trick.

To recapitulate, the flow of events is as follows:

1. Initially, the Start page is requested to render. The `<t:delegate...` will request the tab object to display
2. Start.java contains a method, getSelectedTab(), which is invoked, and by default returns the component called tab1.)Since all components are the same, I've set the message to tab1 as well to distinguish them).
3. Since the component tab1 is returned, it takes the place in the `<t:block...` statement
4. When the user clicks on tab2, Tab2_Link is invoked, which calls onActionFromTab2_link(). The only thing this does is to set the selectedComponent to 2. **(Question: would it be possible to have an onActionFromTab(int selectedComponent) method instead?)**
5. The page is re-rendered, but since the selectedComponent is 'flashed' to persist, during the next rendering, its value is still 2
6. We now return to the start, but tab2 is returned, and therefore the message will be "tab2".

Notes

Remember that the <t:block> code doesn't have to be located where you would have the instance appear. A component from that block will only appear when a delegate calls it. This is the same as in Tapestry 4.

For example, instead of this:

```
<div id="yui-main">
        <div class="yui-b">
                <div id="mainContent">
                        <t:body/>
                        <t:delegate to="selectedTab"/>
                        <t:block>
                                <div t:id="tab1"/>
                                <div t:id="tab2"/>
                                <div t:id="tab3"/>
                        </t:block>
                </div>
        </div>
</div>
```

You could do this, and it will work the same:

```
<div id="yui-main">
        <div class="yui-b">
                <div id="mainContent">
                        <t:body/>
                        <t:delegate to="selectedTab"/>

                </div>
        </div>
</div>

<t:block>
 <div t:id="tab1"/>
 <div t:id="tab2"/>
 <div t:id="tab3"/>
</t:block>
```

In fact, you could have all the blocks for your page in the same <t:block> tag, and multiple delegates can pull from this pool of blocks.

It may also be possible to inject blocks in your page class, or pull blocks from other pages, if you really wanted to.